«Северо-Кавказский институт бизнеса, инженерных и информационных технологий» (ЧОУ ВО СКИБИИТ)
Кафедра информационных технологий

СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

методические указания по выполнению лабораторных заданий
для бакалавров направления подготовки
09.03.01 - Информатика и вычислительная техника

Армавир
2015

Методические указания предназначены для бакалавров направления подготовки 09.03.01**,** изучающих вычислительные системы. Методические указания содержат сведения, необходимые для правильного выполнения и оформления лабораторных работ.

Составители: Нелин В.М.

Оглавление

Введение

Наше пособие соответствует требованиям проектов ФГОС по направлениям «Информатика и вычислительная техника» и «Информационные системы», ориентированных на подготовку магистров.

В работе над пособием нам всегда помогали студенты. Благодарим их.

Значительную помощь оказали студенты кафедры «Системы обработки информации и управления» (ИУ5) МГТУ им.Н.Э.Баумана разработчики электронных изданий книг по искусственному интеллекту, материал которых включен в печатное пособие и его Студенты кафедры «Информационные технологии» МГУП помогли подобрать литературу и осуществляли анализ Интернет ресурсов в том числе и на иностранных языках по проблематике ИИ, в рамках дипломного проектирования разработали варианты организации и оформления электронных библиотек по различным областям ИИ.

Лабораторная работа №1

Слабые методы решения задач

Цель работы: Научиться формализовывать и решать логические задачи в среде Strawberry Prolog.

Задание: Реализовать логическую задачу с использованием среды Prolog. Введите данные задачи согласно вашему варианту и проверьте работу Вашей программы.

Варианты заданий:

1. Задача из книги Н.Вирта “Алгоритмы + Структуры данных =Программы”.

Я женился на вдове, которая имеет взрослую дочь. Мой отец, который весьма часто навещал нас, влюбился в мою падчерицу и женился на ней. Поэтому мой отец стал моим зятем, а моя падчерица стала моей мачехой. Спустя несколько месяцев моя жена родила сына, который стал шурином (зятем) моему отцу, а потому моим дядей. Жена моего отца, т.е. моя падчерица, тоже родила сына.

Я сам себе дедушка?

2. Задача из книги Дж.Ф.Люгера “Искусственный интеллект. Стратегии и методы решения сложных проблем” о хорошей собаке Фреде и его хозяине Сэме.

Факты и правила в этой задаче представлены предложениями:

а) Фред – это колли.

б) Сэм – хозяин Фреда.

в) Сегодня суббота.

г) В субботу холодно.

д) Фред дрессированный.

е) Спаниели – хорошие собаки, а колли такие же дрессированные.

ж) Если собака хорошая и имеет хозяина, то она находится рядом с хозяином.

з) Если в субботу тепло, то Сэм находится в парке.

и) Если в субботу не тепло, то Сэм находится в музее.

Где Фред?

3. Задача о “счастливом студенте”.

Любой студент, который сдает экзамен по истории и выигрывает в лотерею, - счастлив. Известно, что любой удачливый или старательный студент может сдать все экзамены. Джон не относится к числу старательных студентов, но достаточно удачлив. Любой удачливый студент выигрывает в лотерею. **Счастлив ли Джон?**

4. У героини комедии Шекспира «Венецианский купец» Порции было три шкатулки из золота, серебра и свинца. В одной из шкатулок хранился портрет Порции. На крышках каждой шкатулки были сделаны надписи:

- на золотой: «Портрет в этой шкатулке»;
- на серебряной: «Портрет не в этой шкатулке»;
- на свинцовой: «Портрет не в золотой шкатулке».

Порция объяснила, что среди этих высказываний истинно только одно.

В какой шкатулке портрет Порции?

5. Встретились три подруги: Белова, Краснова и Чернова. На одной из них было черное платье, на другой – красное, на третьей – белое. Девочка в белом платье говорит Черновой: «Нам надо поменяться платьями, а то цвет наших платьев не соответствует фамилиям». **Кто, в какое платье был одет?**

6. В кафе встретились три друга: скульптор Белов, скрипач Чернов и художник Рыжов. «Замечательно, что у одного из нас белые, у другого черные, а у третьего рыжие волосы, но ни у кого цвет волос не соответствует фамилии», заметил черноволосый. «Ты прав», - сказал Белов. **Какой цвет волос у художника?**

7. Коля, Боря, Вова и Юра заняли первые четыре места в соревновании. На вопрос, какие места они заняли, трое из них ответили:

1) Коля ни первое, ни четвертое;
2) Боря второе;
3) Вова не был последним.

Какое место занял каждый мальчик?

8. Ваня, Петя, Саша и Коля носят фамилии, начинающиеся на буквы В, П, С и К. Известно, что : 1) Ваня и С. – отличники; 2) Петя и В. – троечники; 3) В. ростом выше П.; 4) Коля ростом ниже П.; 5) Саша и Петя имеют одинаковый рост. **На какую букву начинается фамилия каждого мальчика?**

9. Король приказал своим солдатам сыскать пропавшее варенье, и оно было найдено в домике, где обитали Мартовский Заяц, Болванщик и Соня. Разумеется, все трое были схвачены и предстали перед судом.

- Я требую, - заявил Король, обращаясь к судье и присяжным, - чтобы вы до конца разобрались в этом деле.
- Не вы ли случайно украли варенье? - спросил Король у Мартовского Зайца.
- Не крал я никакого варенья! - взмолился Мартовский Зая
- Ну а что скажете вы? - прорычал Король, обращаясь к Болванщику, который дрожал как осиновый лист. - Вы случайно не злоумышленник, который украл варенье?

Болванщик не мог вымолвить ни слова: он только глоток за глотком отпивал свой чай.

- Раз ему нечего сказать, то это доказывает его виновность, - заметила Королева. - Отрубить ему голову!
- Нет, нет! - едва выговорил дрожащим голосом Болванщик. - Варенье украл один из нас, но не я!
- Запишите! - приказал Король присяжным. - Это показание может оказаться очень важным!
- Ну а вы? - продолжал Король, обращаясь к Соне. - Что вы скажете нам обо всем этом? Говорят ли оба ваших соседа. Мартовский Заяц и Болванщик, правду?
- По крайней мере один из них сказал правду, - ответила Соня и мгновенно заснула, да так и проспала до конца судебного заседания.

Как показало расследование, ни Мартовский Заяц, ни Болванщик не сказали правды одновременно. **Кто украл варенье?**

10. У героини комедии Шекспира «Венецианский купец» Порции было три шкатулки из золота, серебра и свинца. В одной из шкатулок хранился портрет Порции. На крышках каждой шкатулки были сделаны надписи:

- на золотой: «Портрет не в серебряной шкатулке»;
- на серебряной: «Портрет не в этой шкатулке»;
- на свинцовой: «Портрет в этой шкатулке».

Порция пояснила, что из трех высказываний по крайней мере одно истинно и по крайней мере одно ложно.

В какой шкатулке ее портрет?

11. У Фрэнка Стоктона есть сказка «Принцесса или тигр ?», в которой узник должен отгадать, в какой из двух комнат находится принцесса, а в какой – тигр. Если он отгадает где принцесса, то станет свободным, иначе – его растерзает тигр.

Узнику объяснили, что в каждой из комнат будет находиться либо принцесса, либо тигр, хотя вполне может статься, что сразу в обеих комнатах обнаружится по тигру или там окажутся одни принцессы.

Табличка на двери 1й комнаты: «В этой комнате находится принцесса, а в другой комнате сидит тигр».

Табличка на двери 2й комнаты: «В одной из этих комнат находится принцесса; кроме того, в одной из этих комнат сидит тигр».

Известно, что одна надпись истинна, а другая ложна. **В какой комнате принцесса?**

12. На некотором острове живут «рыцари», которые всегда горят правду, «лжецы», которые всегда лгут, и нормальные люди, которые иногда лгут, иногда говорят правду.

Трое жителей острова A,B и C, среди которых есть рыцарь, лжец и нормальный человек, высказывают следующие утверждения:

1) A говорит: «Я нормальный человек».
2) B говорит: «Это правда».
3) C говорит: «Я не нормальный человек».

Кто же такие A,B и C?

13. На некотором острове обитают «рыцари», которые всегда говорят правду, и «лжецы», которые всегда лгут. Кроме того в местном лесу водятся оборотни, превращающиеся в волков и пожирающие людей. Оборотень может быть либо «рыцарем», либо «лжецом». Вы берете интервью у трех обитателей острова A, B и C. Известно, что каждый из них либо «рыцарь», либо «лжец» и среди них есть ровно один оборотень.

A и B говорят одно и то же: «Я оборотень», а C утверждает: «Не более чем один из нас «рыцарь».

Проведите полную классификацию A,B и C.

14. На некотором острове обитают «рыцари», которые всегда говорят правду, и «лжецы», которые всегда лгут. Кроме того в местном лесу водятся оборотни, превращающиеся в волков и пожирающие людей. Оборотень может быть либо «рыцарем», либо «лжецом». Вы встретили на острове трех местных жителей A,B и C. Известно, что один из них оборотень и что он «рыцарь», а два остальных жителя – «лжецы». Заявление сделал только B: «C – оборотень».

Кто оборотень?

15. Перед нами три островитянина A,B и C, о каждом из которых известно, что он либо «рыцарь», который всегда говорит правду, либо лжец, который всегда лжет. Условимся называть двух островитян однотипными, если они оба «рыцари» или оба «лжецы».

Пусть A и B высказывают следующие утверждения:

- A: «B – лжец»
- B: «A и C однотипны».

Кто такой C: «рыцарь» или «лжец»?

16. В некотором месте есть только три деревни: Правдино, Кривдино и Серединка-Наполовинку. Соответственно, жители первой всегда говорят правду, жители второй - всегда лгут, а в Серединке-Наполовинку часть жителей всегда честные, часть - всегда лгуны. Вы - пожарник, который сидит в пожарном участке, откуда этих трех деревень не видно. Раздается телефонный звонок, Вы берете трубку:

- У нас в деревне пожар.
- А где вы живете?
- В Серединке-Наполовинку.

Спрашивается: куда ехать?

17. Трое ребят вышли гулять с собакой, кошкой и хомячком. Известно, что Петя не любит кошек и живет в одном подъезде с хозяйкой хомячка. Лена дружит с Таней, гуляющей с кошкой. **Определить, с каким животным гулял каждый из детей.**

18. Пятеро студентов едут на велосипедах. Их зовут Сергей, Борис, Леонид, Григорий и Виктор. Велосипеды сделаны в пяти городах: Риге, Пензе, Львове, Харькове и Москве. Каждый из студентов родился в одном из этих городов, но ни один студент не едет на велосипеде, сделанном на его родине. Сергей едет на велосипеде, сделанном в Риге. Борис родом из Риги, у него велосипед из Пензы. У Виктора велосипед из Москвы. У Григория велосипед из Харькова. Виктор родом из Львова. Уроженец Пензы едет на велосипеде, сделанном на родине Леонида. **Кто из студентов родом из Москвы?**

19. В велосипедных гонках три первых места заняли Алеша, Петя и Коля. **Какое место занял каждый из них**, если Петя занял не второе и не третье место, а Коля – не третье?

Содержание отчета:

1. цель работы;
2. задание;
3. схема Вашей продукционной системы.
4. блок-схемы алгоритма унификации и алгоритма поиска.
5. описание тестового прогона (последовательность применения продукций).
6. выводы по работе.

Краткие теоретические сведения

Пролог - это язык программирования, предназначенный для обработки символьной информации. Особенно хорошо он приспособлен для решения задач, в которых фигурируют объекты и отношения между ними.

Одним из замечательных свойств Пролога является то, что это достаточно простой язык, и студенты могли бы использовать его непосредственно в процессе изучения вводного курса по искусственному интеллекту.

Рассмотрим некоторые особенности языка Пролог.

- Программирование на Прологе состоит в определении отношений и в постановке вопросов, касающихся этих отношений
- Пролог-программа состоит из предложений. Каждое предложение заканчивается **точкой.**
- Прологовские предложения бывают трех типов: факты, правила и вопросы.

Факты содержат утверждения, которые являются всегда, безусловно верными.

Правила содержат утверждения, истинность которых зависит от некоторых условий.

С помощью **вопросов** пользователь может спрашивать систему о том, какие утверждения являются истинными. Чтобы получить вопрос необходимо перед предложением поставить символы: **?-.**

- Предложения Пролога состоят из головы и тела. **Тело** - это список целей, разделенных запятыми или точкой запятой.
- Последовательность целей, перечисляемая через:

, означает конъюнкцию этих целевых утверждений (логическое И);

; означает дизъюнкцию этих целевых утверждений (логическое ИЛИ).

Факты - это предложения, имеющие пустое тело. **Вопросы** имеют только тело. **Правила** имеют голову и (непустое) тело. В правилах голова и тело разделены символами **:-.**

- Аргументы отношения могут быть: конкретными объектами (**атомами**) или абстрактными объектами (**переменными**). **Атомы м**огут представлять собой цепочки следующих символов:

* прописные буквы A, B, ..., Z
* строчные буквы a, b, ..., z
* цифры 0, 1, 2, ..., 9
* специальные символы, такие как

*** + - * / = : . & _ ~**

Атомы можно создавать тремя способами:

(1) из цепочки букв, цифр и символа подчеркивания _, начиная такую цепочку со строчной буквы:

aHCa
x25
x_

(2) из специальных символов:

<--->
======>
...
.
...
: : =

Следует соблюдать некоторую осторожность, поскольку часть цепочек специальных символов имеют в Прологе заранее определенный смысл. Примером может служить :- .

(3) из цепочки символов, заключенной в одинарные кавычки. Это удобно, если мы хотим, например, иметь атом, начинающийся с прописной буквы. Заключая его в кавычки, мы делаем его отличным от переменной:

'АНСа'

Переменные - это цепочки, состоящие из букв, цифр и символов подчеркивания. Они начинаются с прописной буквы или с символа подчеркивания:

_X

X

Result

Если переменная встречается в предложения только один раз, то нет необходимости изобретать ей имя. Можно использовать так называемую "анонимную" переменную, которая записывается в виде одного символа подчеркивания.

- Пролог-система рассматривает вопросы как цели, к достижению которых нужно стремиться. Ответ на вопрос может оказаться или положительным или отрицательным в зависимости от того, может ли быть соответствующая цель достигнута или нет. Если на вопрос существует несколько ответов, пролог-система найдет столько из них, сколько пожелает пользователь (в Strawberry Prolog возможно получение очередного ответа нажатием клавиши F8).
- По ходу вычислений вместо переменной может быть подставлен другой объект. Мы говорим в этом случае, что переменная **конкретизирована.**

Разберем следующую задачу об “интересной жизни”.

Все небедные и умные люди счастливы. Человек, читающий книги, - неглуп. Джон умеет читать и является состоятельным человеком. Счастливые люди живут интересной жизнью. Можно ли указать человека, живущего интересной жизнью?

На языке Prolog эту задачу можно записать следующим образом:

```
happy(X):-rich(X), clever(X).
clever(X):-read(X).
rich(tom).
read(tom).
interesting_life(X):-happy(X).
?-interesting_life(X), write(X).
```

Это программа содержит 6 предложений- 3 правила, 2 факта и вопрос. Результатом работы данной Пролог-программы будет ответ системы: tomYes. То есть решением задачи будет ответ: человек, живущий интересной жизнью – это Том.

Лабораторная работа №2

Сильные методы решения задач

Цель работы:

Изучение основных правил представления знаний в экспертных системах. Получение навыков разработки экспертной системы.

Задание:

При выполнении лабораторной работы используется программа **Mini Expert System** («Малая экспертная система» вер. 1.0), описание работы с которой приведено ниже.

В соответствии с вариантом составить список вопросов, необходимых для получения определенного решения и список вариантов ответов. Списки оформить в виде текстового файла с расширением .DAT (см. примеры из программы Mini Expert System). Отладить экспертную систему, проверить ее работоспособность на примерах.

Содержание отчета:

1. цель работы;
2 задание;
3. списки вопросов и вариантов ответов (распечатка .DAT-файла);
4. протоколы проверки работоспособности на примерах;
5. выводы по работе.

Варианты заданий:

1. Определить цветок по заданным характеристикам. БЗ содержит сведения о призна- ках цветов.
2. Определить наличие вирусов в компьютере по характерным их проявлениям, если таковые наблюдаются. БЗ содержит сведения о характерных проявлениях вирусов, определении принадлежности вируса какому-то классу вирусов, и другие сведения.
3. Определение болезни человека по характерным симптомам. БЗ содержит сведения о проявлениях различных заболеваний и о самих болезнях, факты отражают текущее состояние человека.
4. Определение знака зодиака человека.
5. Определение животного (можно мифического, сказочного) и соотнесение его определенному классу животных по внешним признакам.
6. Определение дерева по заданным его признакам (например, листья широкие, узкие, игольчатые; кора светлая, темная; крона широкая и низкая, узкая и высокая, плотная, редкая и др.).
7. Определение темперамента личности по ее проявлениям. В РП содержатся факты, характеризующие поведение личности (например, вспыльчивый, быстрая перемена настроения, громкий голос). По правилам в БЗ нужно сделать вывод, к какой группе (холерик, сангвиник, флегматик, меланхолик) принадлежит личность (например, «Если голос тихий и активность низкая то группа = меланхолик»).
8. Определение породы собаки по характерным внешним признакам.

9. Определение жанра кинофильма по каким-либо признакам. БЗ содержит такие пра- вила. На основании актерского состава, сцен фильма, музыки определяется принадлеж- ность фильма к какому-либо жанру.
10. Определить тип литературного произведения по его признакам (например, наличие стихотворного ритма, текст с разделением на строки, длина произведения малая).
11. Определить по заданным признакам компьютерной игры, что это за игра (или тип игры).
12. Определение профессии человека по заданным признакам его работы.
13. Определение причины неисправности компьютера.
14. Определение эры (периода) Земли по разнообразию растений и животных на плане- те. Например, если среди животных встречаются динозавры, то в данный момент – эра мезозоя.
15. Определение страны (государства) по признакам его флага, герба. Задаются факты (например, элементы построения флага – только горизонтальные полосы, количество цветов флага, порядок размещения полос и т.д.), программа, используя их и БЗ, выводит заключение о том, что за страна имеет данный флаг (герб).
16. Определение причины неисправности автомобиля по заданным отклонениям в его работе.
17. Определение расы человека по заданным характеристикам его внешности (напри- мер, европеоидная раса, монголоидная раса и т.д.).
18. Определение вида транспорта по его внешним признакам и значениям определенных параметров.

Краткие теоретические сведения

1. Структура экспертной системы

Под **экспертной системой** (ЭС) понимают набор программ, выполняющий функции эксперта при решении задач из некоторой предметной области. ЭС выдают советы, проводят анализ, дают консультации, ставят диагноз.

Главным достоинством ЭС, определяющим сравнительно высокий интерес к ним как к методам искусственного интеллекта, является возможность накопления знаний и сохранение их длительное время.

Типичная статическая ЭС состоит из следующих основных компонентов:
- подсистемы логического вывода (интерпретатора, решателя);
- рабочей памяти (РП), называемой также базой данных (БД);
- базы знаний (БЗ);
- компонента приобретения знаний;
- подсистемы объяснения решений;
- интерфейса пользователя.

Обобщенная схема статической *ЭС* приведена на рисунке 1.

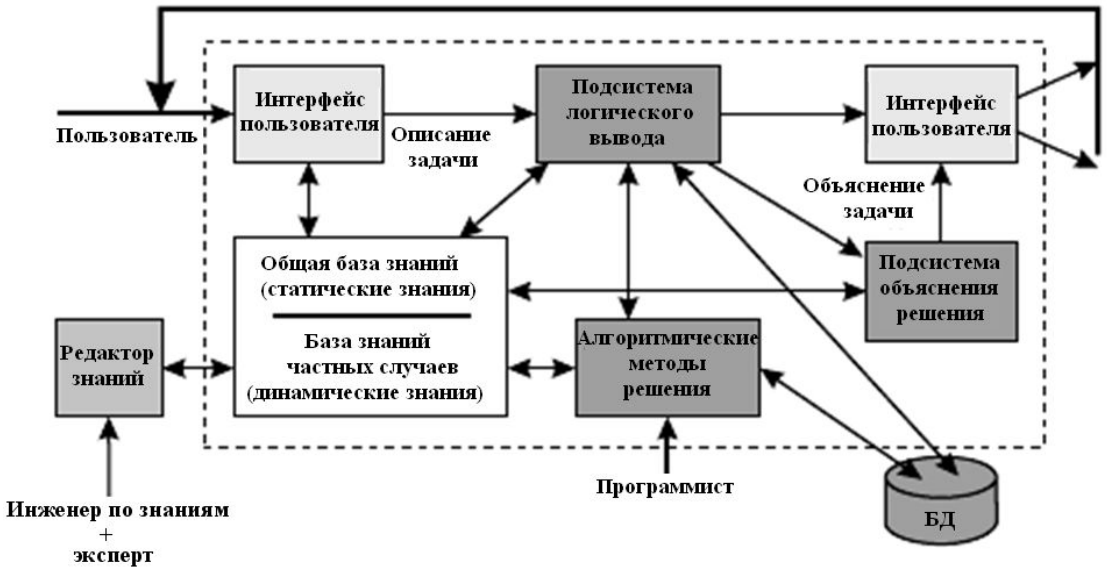

Рис. 1. Структура статической экспертной системы

База данных (рабочая память) предназначена для хранения исходных и промежуточных данных решаемой в текущий момент задачи.

База знаний — ядро экспертной системы, предназначена для хранения долгосрочных данных, описывающих рассматриваемую область, и правил, описывающих преобразования данных этой области. Чаще всего для представления знаний в *ЭС* используются системы продукций и семантические сети.

Пользователь — специалист предметной области, для которого предназначена система. Обычно его квалификация недостаточно высока, и поэтому он нуждается в помощи и поддержке со стороны экспертной системы.

Интерфейс пользователя — комплекс программ, ориентированный на организацию дружественного общения с пользователем в ходе решения задач, в процессе приобретения знаний и объяснения результатов работы.

Подсистема логического вывода (машина вывода, решатель, интерпретатор)— программа, моделирующая ход рассуждений эксперта на основании исходных данных из рабочей памяти и знаний, имеющихся в базе знаний.

Модуль объяснения решений — объясняет, как система получила решение задачи (или почему она не получила решение) и какие знания она при этом использовала, что облегчает эксперту тестирование системы и повышает доверие пользователя к полученному результату.

Ответ на вопрос «Как?» — это трассировка всего процесса получения решения с указанием использованных фрагментов базы знаний, то есть всех шагов цепи умозаключения. Ответ на вопрос «Почему?» — ссылка на умозаключение, непосредственно предшествовавшее полученному решению, то есть отход на один шаг назад.

Интеллектуальный редактор базы знаний — программа, автоматизирующая процесс наполнения ЭС знаниями, осуществляемый инженером по знаниям.

Система функционирует в следующем циклическом режиме:

1. Диалоговый компонент запрашивает данные или результатов анализов, наблюдений (этот этап может быть реализован в виде системы вопросов к пользователю) и помещает их в рабочую память (базу данных).

2. Подсистема логического вывода интерпретирует результаты с помощью правил, извлеченных из базы знаний.
3. В случае нехватки информации для окончательного решения процесс продолжается до тех пор, пока не поступит достаточное количество информации.

Все перечисленные выше знания хранятся в базе знаний. Для ее построения требуется провести опрос специалистов, являющихся экспертами в конкретной предметной области.

2. Продукционные системы

База знаний - наиболее важная компонента экспертной системы, на которой основаны ее «интеллектуальные способности». Существует несколько способов представления знаний в ЭС, однако общим для всех них является то, что знания представлены в символьной форме (элементарными компонентами представления знаний являются тексты, списки и другие символьные структуры). Тем самым в ЭС реализуется принцип символьной природы рассуждений, который заключается в том, что процесс рассуждения представляется как последовательность символьных преобразований.

Наиболее простым с точки зрения построения и широко используемым типом моделей принятия решений являются **продукционные системы** (ПС). Они представляют собой структурированные наборы **продукционных правил** (ПП) вида

$$PR = <S, N, F, A \Rightarrow C, W>,$$

где S - сфера применения данного правила; N - номер или имя правила; F - условие активизации данного правила; $A \Rightarrow C$ - ядро ПП; W - постусловие.

В состав правил могут входить условия активизации F, которые представляют собой либо переменную, либо логическое выражение (предикат). Когда F принимает значение «истина», ядро продукции может быть активизировано. Если F «ложно», то ядро не активизируется.

Постусловие W описывает, какие изменения следует внести в ПС, и актуализируется только после того, как ядро продукции реализовалось.

Интерпретация ядра может быть различной в зависимости от вида A и C, находящихся по разные стороны знака секвенции «⇒».Наиболее часто в ПС используют ПП вида

«если A, то C»,

где A и C - логические выражения, которые могут включать в себя другие выражения;

C может выполняться с определенной вероятностью:

«если A, то возможно C».

Возможность может определяться некоторыми оценками реализации ядра. Например, если задана вероятность выполнения C при актуализации A, то ПП может быть таким:

«если A, то с вероятностью P выполнить C».

Оценка реализации ядра может быть лингвистической, связанной с лингвистической переменной:

«если A, то с большей долей уверенности возможно C».

Прогнозирующие ПП, в которых описываются, например, последствия, ожидаемые при актуализации А:

«если А, то с вероятностью Р можно ожидать С».

Достоинствами ПС являются:

- удобство описания процесса принятия решения экспертом (формализация его интуиции и опыта);
- простота редактирования модели;
- прозрачность структуры.

ПС в качестве моделей применимы в следующих случаях:

- не могут быть построены строгие алгоритмы или процедуры принятия решений, но существуют эвристические методы решения;
- существует, по крайней мере, один эксперт, который способен явно сформулировать свои знания и объяснить свои методы применения этих знаний при принятии решения;
- пространство возможных решений относительно невелико (число решений счетно);
- задачи решаются методом формальных рассуждений;
- данные и знания надежны и не изменяются со временем.

3. Разработка экспертной системы в программе Mini Expert System

3.1. Подготовка базы знаний

Программа Mini Expert System представляет собой простую экспертную систему, использующую байесовскую систему логического вывода. Она предназначена для проведения консультации с пользователем в какой-либо прикладной области (на которую настроена загруженная база знаний) с целью определения вероятностей возможных исходов и использует для этого оценку правдоподобности некоторых предпосылок, получаемую от пользователя.

На первом этапе создания базы знаний необходимо сформулировать знания о рассматриваемой области в виде двух наборов: Q = {q_j} – набор вопросов (симптомов, свидетельств) и V = {v_i} - набор вариантов исхода (вариантов решения), а также двух матриц вероятностей: Py = {py_{ij}} и Pn = {pn_{ij}} размером m×n, где py_{ij} – вероятность получения положительного ответа на j-й вопрос, если i-й исход верен; pn_{ij} – вероятность получения отрицательного ответа на j-й вопрос, если i-й исход верен; n и m – количества вопросов и исходов соответственно. Кроме того, каждому исходу ставится в соответствие априорная вероятность данного исхода P, т.е. вероятность исхода в случае отсутствия дополнительной информации.

В процессе работы ЭС решатель, пользуясь данными наборами и матрицами и теоремой Байеса, определяет апостериорную вероятность каждого исхода, то есть вероятность, скорректированную в соответствии с ответом пользователя на каждый вопрос:

- при положительном ответе $P_{апостер.} = \frac{Py_{ij} \times P_i}{Py_{ij} \times P_i + Pn_{ij} \times (1 - P_i)}$,
- при отрицательном ответе $P_{апостер.} = \frac{(1 - Py_{ij}) \times P_i}{(1 - Py_{ij}) \times P_i + (1 - Pn_{ij}) \times (1 - P_i)}$,
- при ответе «не знаю» апостериорная вероятность равна априорной.

То есть вероятность осуществления некой гипотезы при наличии определенных подтверждающих свидетельств вычисляется на основе априорной вероятности этой гипотезы

без подтверждающих свидетельств и вероятностей осуществления свидетельств при условиях, что гипотеза верна или неверна.

Исходная информация оформляется в виде текстового файла с расширением .DAT со следующей структурой:

Описание базы знаний, имя автора, комментарий и т.д.
(можно в несколько строк; эта информация выводится после загрузки базы знаний; данная секция заканчивается после первой пустой строки)

Вопрос № 0 (любой текст, заканчивающийся переносом строки)
Вопрос № 1
Вопрос № 2
...
Вопрос № N (после последнего вопроса следует одна пустая строка, и вторая секция заканчивается)

Исход № 0, P [, i, Py, Pn]
Исход № 1, P [, i, Py, Pn]
Исход № 2, P [, i, Py, Pn]
...
Исход № M, P [, i, Py, Pn]

В последней секции перечисляются исходы и соответствующие им элементы матриц вероятностей. Каждый исход задаётся в отдельной строке, перечисление заканчивается с концом файла.

В начале описания правила вывода задаётся исход, вероятность которого меняется в соответствии с данным правилом. Это текст, включающий любые символы, кроме запятых. После запятой указывается априорная вероятность данного исхода P. После этого через запятую идёт ряд повторяющихся полей из трёх элементов. Первый элемент i – номер соответствующего вопроса. Следующие два элемента Py_{ij} и Pn_{ij} – соответственно вероятности получения ответа «Да» на этот вопрос, если возможный исход верен и неверен. Эти данные указываются для каждого вопроса, связанного с данным исходом.

Примечание: P <= 0.00001 считается равной нулю, а P >= 0.99999 – единице, поэтому не следует указывать такие значения – исход с подобной априорной вероятностью обрабатываться не будет.

Например:

Грипп, 0.01, 1,0.9,0.01, 2,1,0.01, 3,0,0.01

Здесь сказано: существует априорная вероятность P = 0,01 того, что любой наугад взятый человек болеет гриппом.

Первому вопросу (i = 1) соответствует запись «1,0.9,0.01». Отсюда следуют значения $Py_{i1} = 0,9$ и $Pn_{i1} = 0,01$, которые означают, что если у пациента грипп, то он в девяти случаях из десяти ответит «Да» на этот вопрос, а если у него нет гриппа, он ответит «Да» лишь в одном случае из ста (т.е. данный симптом встречается довольно редко при других болезнях). Ответ «Да» подтверждает гипотезу о том, что у него грипп. Ответ «Нет» позволяет предположить, что человек гриппом не болеет.

При положительном ответе «Да» (+5) на первый вопрос апостериорная вероятность для рассматриваемого примера составит:

$$P_{апостер.} = \frac{Py_{ij} \times P_i}{Py_{ij} \times P_i + Pn_{ij} \times (1 - P_i)} = \frac{0.01 * 0.9}{0.01 * 0.9 + (1 - 0.01) * 0.01} = 0{,}47619.$$

При отрицательном ответе «Нет» (-5) на первый вопрос апостериорная вероятность для рассматриваемого примера составит:

$$P_{апостер.} = \frac{(1 - Py_{ij}) \times P_i}{(1 - Py_{ij}) \times P_i + (1 - Pn_{ij}) \times (1 - P_i)} =$$

$$= \frac{(1 - 0.9) * 0.01}{(1 - 0.9) * 0.01 + (1 - 0.9) * (1 - 0.01)} = 0{,}00102.$$

При ответе «Не знаю» (0) апостериорная вероятность исхода равна априорной: $P_{апостер.} = P_i$.

При промежуточном ответе *h* (от -5 до 0 и от 0 до +5) апостериорная вероятность рассчитывается с учетом степени уверенности принадлежности признака и рассчитывается линейной интерполяцией от значений утвердительных ответов «Да», «Нет», «Не знаю».

При отрицательном ответе (-5;0):

$$P_{апостер.} = P_i + (P_{апостер.}(неЕ)) \times \frac{h}{5}.$$

Например, при ответе *h* = -3:

$$P_{апостер.} = 0.01 + (0.01 - 0.00102) \times \frac{-3}{5} = 0{,}00461.$$

При отрицательном ответе (0;+5):

$$P_{апостер.} = P_i + (P_{апостер.}(Е) - P_i) \times \frac{h}{5}.$$

Например, при ответе *h* = +3:

$$P_{апостер.} = 0.01 + (0.47619 - 0.01) \times \frac{3}{5} = 0{,}28971.$$

Для второго вопроса имеем запись «2,1,0.01». То есть, если у человека грипп, то этот симптом обязательно должен присутствовать (Py_{i2} = 1) и он обязательно ответит «Да». Соответствующий симптом может иметь место и при отсутствии гриппа (Pn_{i2} = 0,01), но это маловероятно.

Примечание: При большом количестве вопросов нет необходимости в каждой строке последней секции перечислять их все, тем более если ответ на какой-либо вопрос не влияет на вероятность данного исхода.

3.2. Работа с программой Mini Expert System

MiniES v1.0 - "малая экспертная система" версия 1.0 представляет собой простую экспертную систему, использующую байесовскую систему логического вывода. Она предназначена для проведения консультации с пользователем в какой-либо прикладной

области (на которую настроена загруженная база знаний) с целью определения вероятностей возможных исходов и использует для этого оценку правдоподобности некоторых предпосылок, получаемую от пользователя. Важным достоинством данной программы является возможность создания и использования собственной базы знаний.

Запускающим файлом программы является MiniES.exe. После запуска появляется диалоговое окно (рисунок 2), кнопки на котором выполняют функции: «Загрузить базу знаний» - загрузка заранее подготовленного .DAT-файла; «Начать консультацию» - запуск решателя, ответы на задаваемые вопросы вводятся в нижнее поле по шкале от -5 (однозначно нет) до 5 (однозначно да).

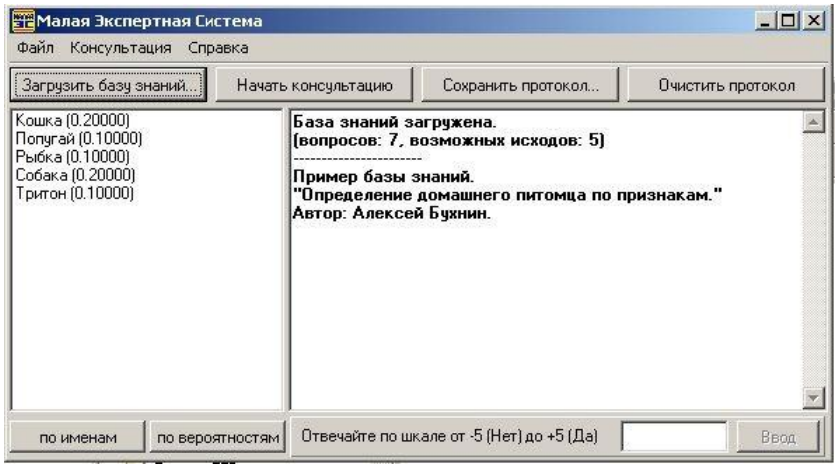

Рис. 2. Диалоговое окно программы Mini Expert System

Вопросы на защиту:

1. Дайте определение термину «экспертная система».
2. Задачи, решаемые экспертной системой?
3. Классификация экспертных систем?
4. Структурная схема статической экспертной системы, назначение отдельных блоков.
5. Структурная схема динамической экспертной системы?
6. В чем отличие статической и динамической экспертных систем?
7. Основные режимы работы экспертных систем?
8. Технология разработки экспертной системы: дать краткую характеристику основным этапам.

Лабораторная работа №3

Обучение нейронных сетей

Цель работы: Ознакомиться с особенностями построения и обучения нейронных сетей. Получить навыки работы с пакетом NNTool MatLab.

Задание: Реализовать нейронную сеть прямого распространения с одним нейроном в входном слое, N нейронами во скрытом слое с сигмоидальной функцией активации (*logsig*) и одним нейроном в выходном слое с линейной функцией активации. Обучить ее аппроксимации заданной функции по алгоритму обратного распространения. Выполнить сравнение качества аппроксимации для полиномиальной, синусоидальной и *humps* функций. Для одной из функций выполнить сравнение качества аппроксимации для различных N.

Варианты заданий:

1. Алгоритм наискорейшего спуска
2. Алгоритм градиентного спуска с рестартами Пауэлла-Биля
3. Алгоритм градиентного спуска с обновлением по Флетчеру-Ривсу
4. Алгоритм градиентного спуска с обновлением по Полаку-Рибьеру
5. Адаптивный алгоритм градиентного спуска
6. Алгоритм градиентного спуска с моментом инерции
7. Алгоритм градиентного спуска с одношаговой секущей
8. Алгоритм переменной метрики с вычислением гессиана по формуле Бройдена-Флетчера-Гольдфарба-Шенно (BFGS)
9. Алгоритм переменной метрики с вычислением гессиана по формуле Девидона-Флетчера-Пауэлла (DFP)
10. Алгоритм Левенберга-Марквардта
11. Алгоритм Quickprop
12. Модифицированный алгоритм Quickprop
13. Алгоритм RPROP

Примечание: Желательно выполнить данную лабораторную работу в среде Matlab.

Содержание отчета:

1. цель работы
2. задание
3. описание алгоритма.
4. сравнение работы алгоритма для 3-х функций разного типа.
5. графики, иллюстрирующие качество аппроксимации для различных N.
6. выводы по работе.

Краткие теоретические сведения

Нейронные сети (НС) широко используются для решения разнообразных задач. Основы теории и технологии применения НС широко представлены в пакете MATLAB графическим интерфейсом пользователя NNTool.

Чтобы запустить NNTool, необходимо выполнить команду в командном окне MATLAB:

>> nntool

после этого появится главное окно NNTool, именуемое **"Окном управления сетями и данными" (Network/Data Manager)** (рис. 1).

Рис.1. Главное окно NNTool

Панель "Сети и данные" (Networks and Data) имеет функциональные клавиши со следующими назначениями:

Помощь (Help)- краткое описание управляющих элементов данного окна;

Новые данные (New Data)- вызов окна, позволяющего создавать новые наборы данных;

Новая сеть (New Network)- вызов окна создания новой сети;

Импорт (Import)- импорт данных из рабочего пространства MATLAB в пространство переменных HCTool;

Экспорт (Export)- экспорт данных из пространства переменных NNTool в рабочее пространство MATLAB;

Вид (View)- графическое отображение архитектуры выбранной сети;

Удалить (Delete)- удаление выбранного объекта.

На панели **"Только сети" (Networks only)** расположены клавиши для работы исключительно с сетями. При выборе указателем мыши объекта любого другого типа, эти кнопки становятся неактивными.

При работе с NNTool важно помнить, что клавиши View, Delete, Initialize, Simulate, Train и Adapt (изображены на рис. 1 как неактивные) действуют применительно к тому объекту, который отмечен в данный момент выделением. Если такого объекта нет, либо над выделенным объектом невозможно произвести указанное действие, соответствующая клавиша неактивна.

Одним из самых замечательных свойств нейронных сетей является способность аппроксимировать и, более того, быть универсальными аппроксиматорами. Сказанное означает, что с помощью нейронных цепей можно аппроксимировать сколь угодно точно непрерывные функции многих переменных. Рассмотрим создание нейронной сети с помощью NNTool на примере.

Необходимо выполнить аппроксимацию функции следующего вида:

$$\sin\left(\frac{5\pi x}{N}+\sin\left(\frac{7\pi x}{N}\right)\right),$$

где x Є 1÷N, а N - число точек функции.

Создание сети

Перед созданием сети необходимо заготовить набор обучающих и целевых данных.

Теперь заготовим набор обучающих данных (1, 2, 3, … , 100), задав их следующим выражением: [1:100].

Воспользуемся кнопкой **New Data**. В появившемся окне следует произвести изменения, показанные на рис. 2, и нажать клавишу **"Создать" (Create).**

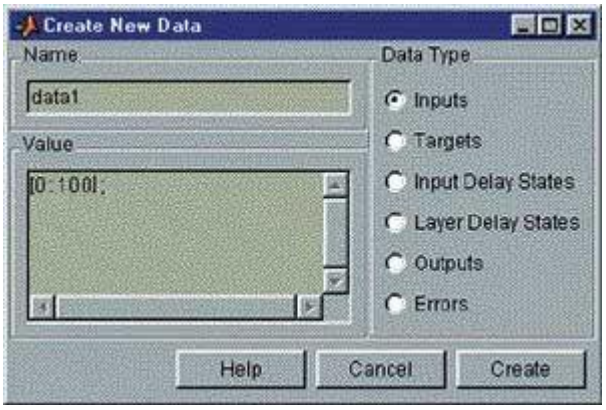

Рис. 2. Задание входных векторов

После этого в окне управления появится вектор **data1** в разделе **Inputs**.

Вектор целей задаётся схожим образом (рис. 3).

Введем в поле **"Значение" (Value)** окна создания новых данных выражение:

```
sin(5*pi*[1:100]/100+sin(7*pi*[1:100]/100)).
```

Эта кривая представляет собой отрезок периодического колебания с частотой 5p/N, модулированного по фазе гармоническим колебанием с частотой 7N (рис. 9).

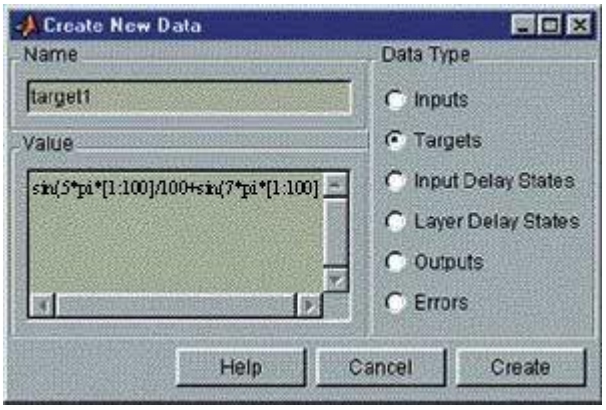

Рис. 3. Задание целевого вектора

После нажатия на **Create** в разделе **Targets** появится вектор **target1**.

Теперь следует приступить к созданию нейронной сети. Выбираем кнопку **New Network** и заполняем форму. Выберем персептрон **(Feed-Forward Back Propagation)** с тринадцатью **сигмоидными (TANSIG)** нейронами скрытого слоя и одним **линейным (PURELIN)** нейроном выходного слоя. Обучение будем производить, используя **алгоритм Левенберга-Маркардта (Levenberg-Marquardt)**, который реализует функция **TRAINLM**. Функция ошибки - MSE.

Поля несут следующие смысловые нагрузки:

- **Имя сети (Network Name)** - это имя объекта создаваемой сети.

- **Тип сети (Network Type)**- определяеттип сети и в контексте выбранного типа представляет для ввода различные параметры в части окна, расположенной ниже этого пункта. Таким образом, для разных типов сетей окно изменяет сво╥ содержание.

- **Входные диапазоны (Input ranges)**- матрица с числом строк, равным числу входов сети. Каждая строка представляет собой вектор с двумя элементами: первый - минимальное значение сигнала, которое будет подано на соответствующий вход сети при обучении, второй - максимальное. Для упрощения ввода этих значений предусмотрен выпадающий список "Получить из входа" (Get from input), позволяющий автоматически сформировать необходимые данные, указав имя входной переменной.

- **Количество нейронов (Number of neurons)**- число нейронов в слое.

- **Передаточная функция (Transfer function)**- в этом пункте выбирается передаточная функция (функция активации) нейронов.

- **Функция обучения (Learning function)**- функция, отвечающая за обновление весов и смещений сети в процессе обучения.

С помощью клавиши **"Вид" (View)** можно посмотреть архитектуру создаваемой сети (рис. 4).

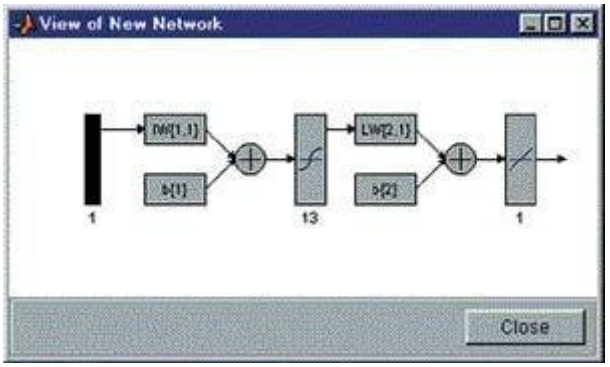

Рис.4. Архитектура сети для решения задачи аппроксимации

Окно предварительного просмотра можно закрыть, нажав клавишу **"Закрыть" (Close)**, и подтвердить намерение создать сеть, нажав **"Создать" (Create)** в окне создания сети.

В результате проделанных операций в разделе **"Сети" (Networks)** главного окна NNTool появится объект с именем **network1.**

Обучение сети

Наша цель - построить нейронную сеть, которая выполняет аппроксимацию заданной функции. Очевидно, нельзя рассчитывать на то, что сразу после этапа создания сети последняя будет обеспечивать правильный результат (правильное соотношение "вход/выход"). Для достижения цели сеть необходимо должным образом обучить, то есть подобрать подходящие значения параметров. В MATLAB реализовано большинство известных алгоритмов обучения нейронных сетей. Создавая сеть, мы указали **TRAINLM** в качестве функции, реализующей алгоритм обучения.

Вернемся в главное окно NNTool. На данном этапе интерес представляет нижняя панель **"Только сети" (Networks only)**. Нажатие любой из клавиш на этой панели вызовет

окно, на множестве вкладок которого представлены параметры сети, необходимые для её обучения и прогона, а также отражающие текущее состояние сети.

Отметив указателем мыши объект сети **network1**, вызовем окно управления сетью нажатием кнопки. Перед нами возникнет вкладка **"Train"** окна свойств сети, содержащая, в свою очередь, еще одну панель вкладок (рис. 5). Их главное назначение - управление процессом обучения. На вкладке **"Информация обучения" (Training info)** требуется указать набор обучающих данных в поле **"Входы" (Inputs)** и набор целевых данных в поле **"Цели" (Targets)**. Поля **"Выходы" (Outputs)** и **"Ошибки" (Errors)** NNTool заполняет автоматически. При этом результаты обучения, к которым относятся выходы и ошибки, будут сохраняться в переменных с указанными именами.

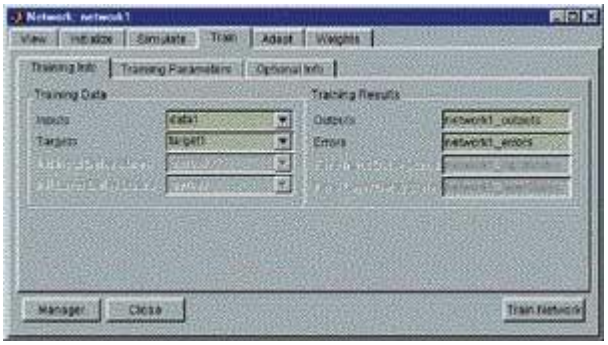

Рис. 5. Окно параметров сети, открытое на вкладке "обучение" (Train)

Завершить процесс обучения можно, руководствуясь разными критериями. Возможны ситуации, когда предпочтительно остановить обучение, полагая достаточным некоторый интервал времени. С другой стороны, объективным критерием является уровень ошибки.

На вкладке **"Параметры обучения" (Training parameters)** для нашей сети (рис. 6) можно установить следующие поля:

Количество эпох (epochs) - определяет число эпох (интервал времени), по прошествии которых обучение будет прекращено. *Эпохой* называют однократное представление всех обучающих входных данных на входы сети.

Достижение цели или попадание (goal) - здесь задаётся абсолютная величина функции ошибки, при которой цель будет считаться достигнутой.

Период обновления (show) - период обновления графика кривой обучения, выраженный числом эпох.

Время обучения (time) - по истечении указанного здесь временного интервала, выраженного в секундах, обучение прекращается.

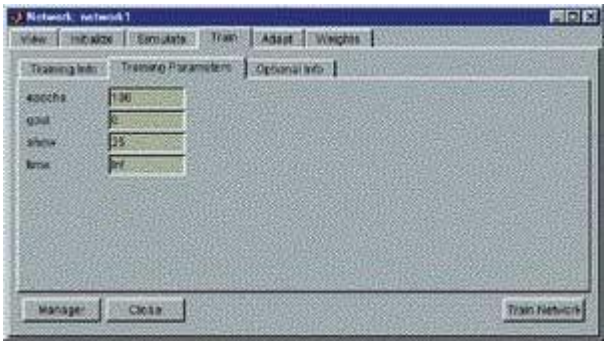

Рис. 6. Вкладка параметров обучения

Следующая вкладка **"Необязательная информация" (Optional Info)** показана на рис. 7.

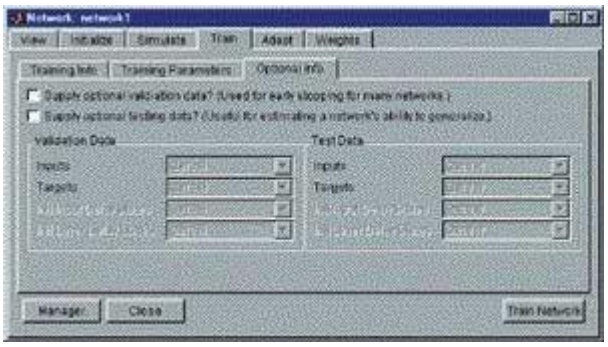

Рис. 7. Вкладка необязательной информации

Чтобы начать обучение, нужно нажать кнопку **"Обучить сеть" (Train Network)** на вкладке **"Обучение" (Train)**. После этого, если в текущий момент сеть не удовлетворяет ни одному из условий, указанных в разделе параметров **обучения (Training Parameters)**, появится окно, иллюстрирующее динамику целевой функции - *кривую обучения*. В нашем случае график может выглядеть так, как показано на рис. 8. Кнопкой **"Остановить обучение" (Stop Training)** можно прекратить этот процесс.

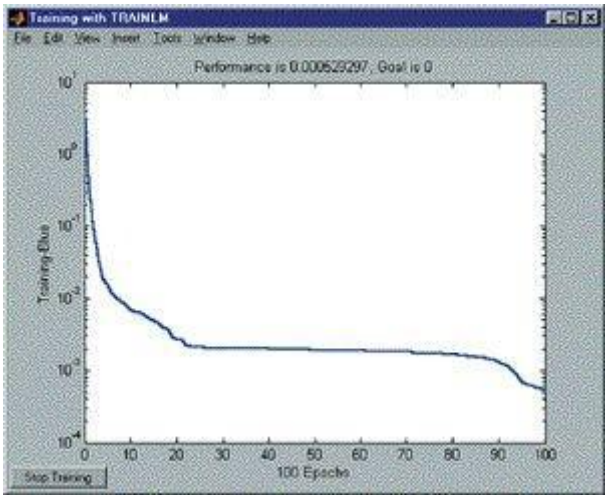

Рис. 8. Кривая обучения в задаче аппроксимации

Из рисунка 8 видно, насколько уменьшилась ошибка аппроксимации за 100 эпох обучения. Форма кривой обучения на последних эпохах говорит также о том, что точность приближения может быть повышена.

Откроем вкладку **"Прогон" (Simulate)** и выберем в выпадающем списке **"Входы" (Inputs)** заготовленные данные. В данной задаче естественно использовать тот же набор данных, что и при обучении **data1.** При желании можно установить флажок "**Задать цели" (Supply Targets)**. Тогда в результате прогона дополнительно будут рассчитаны значения ошибки. Нажатие кнопки **"Прогон сети" (Simulate Network)** запишет результаты прогона в переменную, имя которой указано в поле **"Выходы" (Outputs)**. Теперь можно построить два графика функции в одном окне: первый график – заданная функция $\sin\left(\frac{5\pi x}{N}+\sin\left(\frac{7\pi x}{N}\right)\right)$, второй - в качестве аргумента значение вектора из **Входы" (Inputs),** а в качестве значений функции значение вектора **"Выходы" (Outputs).** Получим следующий график функций

(рис. 9), иллюстрирующий разницу между целевыми данными и полученной аппроксимирующей кривой.

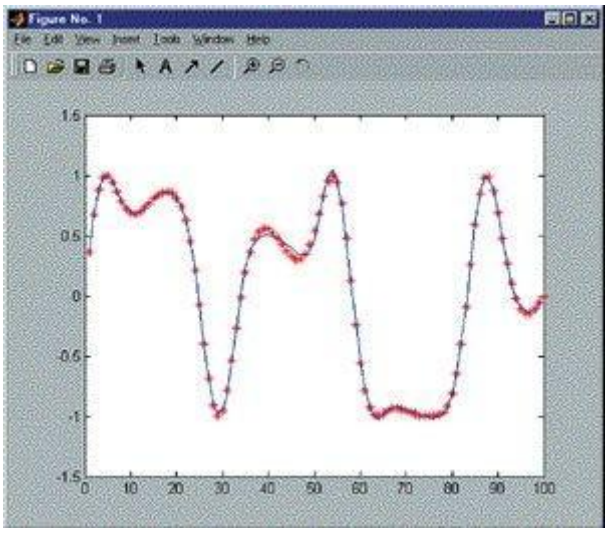

Рис. 9. Красная кривая - целевые данные, синяя кривая - аппроксимирующая функция

Следует заметить, что сеть создается инициализированной, то есть значения весов и смещений задаются определенным образом. Перед каждым следующим опытом обучения обычно начальные условия обновляются, для чего на вкладке **"Инициализация" (Initialize)** предусмотрена функция инициализации. Так, если требуется провести несколько независимых опытов обучения, инициализация весов и смещений перед каждым из них осуществляется нажатием кнопки **"Инициализировать веса" (Initialize Weights)**.

Вернемся к вкладке **"Необязательная информация" (Optional Info)** (рис. 7). Чтобы понять, какой цели служат представленные здесь параметры, необходимо обсудить два понятия: переобучение и обобщение.

При выборе нейронной сети для решения конкретной задачи трудно предсказать её порядок. Если выбрать неоправданно большой порядок, сеть может оказаться слишком гибкой и может представить простую зависимость сложным образом. Это явление называется *переобучением.* В случае сети с недостаточным количеством нейронов, напротив, необходимый уровень ошибки никогда не будет достигнут. Здесь налицо чрезмерное *обобщение*.

Для предупреждения переобучения применяется следующая техника. Данные делятся на два множества: **обучающее (Training Data)** и **контрольное (Validation Data).** Контрольное множество в обучении не используется. В начале работы ошибки сети на обучающем и контрольном множествах будут одинаковыми. По мере того, как сеть обучается, ошибка обучения убывает, и, пока обучение уменьшает действительную функцию ошибки, ошибка на контрольном множестве также будет убывать. Если же контрольная ошибка перестала убывать или даже стала расти, это указывает на то, что обучение следует закончить. Остановка на этом этапе называется **ранней остановкой (Early stopping).**

Таким образом, необходимо провести серию экспериментов с различными сетями, прежде чем будет получена подходящая. При этом чтобы не быть введённым в заблуждение локальными минимумами функции ошибки, следует несколько раз обучать каждую сеть.

Если в результате последовательных шагов обучения и контроля ошибка остаётся недопустимо большой, целесообразно изменить модель нейронной сети (например, услож-

нить сеть, увеличив число нейронов, или использовать сеть другого вида). В такой ситуации рекомендуется применять ещё одно множество - тестовое **множество наблюдений (Test Data)**, которое представляет собой независимую выборку из входных данных. Итоговая модель тестируется на этом множестве, что даёт дополнительную возможность убедиться в достоверности полученных результатов. Очевидно, чтобы сыграть свою роль, тестовое множество должно быть использовано только один раз. Если его использовать для корректировки сети, оно фактически превратится в контрольное множество.

Вопросы на защиту:

1. Структура биологического нейрона.
2. Модель нейрона.
3. Обучение нейронной сети: обучение с учителем.
4. Сети Кохонена.
5. Задача классификации. Алгоритм классификации.
6. Обучение сети Кохонена методом выпуклой комбинации.
7. Применение сети Кохонена для сжатия изображения.

Лабораторная работа №4

Нейросетевой инструментарий среды Matlab

Цель работы: получить навыки работы построения нейронных сетей в среде Matlab с применением пакетов NNTool.

Задание: Решить задачу согласно Вашему варианту в среде Matlab с применением пакетов Neural Network Toolbox.

Варианты заданий:

1. Применить радиальную сеть (Radial basis, newrb) для аппроксимации поверхности peaks (рис. 1) при различных количествах слоев (2-4) и нейронов в слоях (1-30). В отчете привести поверхнось зависимости максимальной ошибки аппроксимации от параметров сети и усредненную поверхность ошибки аппроксимации по всем проведенным экспериментам (поверхность ошибки для каждого эксперимента поделить на величину максимальной ошибки, сложить поверхности для всех экспериментов и поделить на их кол-во). Какая архитектура сети будет оптимальной?

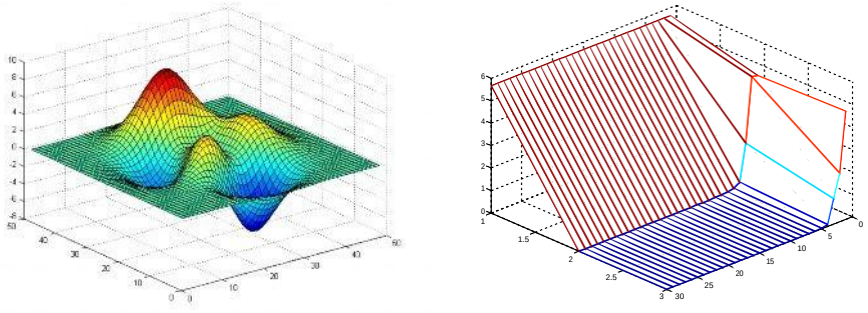

Рис. 1. Поверхность функции двух аргументов peaks (слева) и пример поверхности ошибки аппроксимации в зависимости от параметров сети (справа).

2. Сформировать два класса точек по формулам

```
t = 0:0.01:(2.25*pi); n = length(t);
delta = 70/length(t);
r = 30:delta:(100-delta);
r = exp(r/20)+20;
d = 10;
t1 = t+d1; t2 = t+d2;
x1 = 100+r.*cos(t1)+rand(1,n)*d;
y1 = 100+r.*sin(t1)+rand(1,n)*d;
x2 = 110+r.*cos(t2)+rand(1,n)*d;
y2 = 90+r.*sin(t2)+rand(1,n)*d;
plot(x1, y1, '.r'); hold; plot(x2, y2, '.b');
```

Пример этих классов для `d1 = -pi/4` и `d2 = 3*pi/4` приведен на рис. 2.

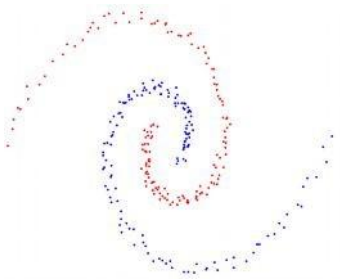

Рис. 2. Пример взаимного расположения двух классов точек – синего и красного.

Построить сеть квантования векторов (Learning vector quantization, newlvq) для разделения этих классов. Сколько нейронов необходимо для этого и как они будут располагаться? Показать траектории нейронов, проходимые ими в процессе обучения.

3. Применить сеть квантования векторов (Learning Vector Quantization, newlvq) для распознавания изображений цифр от 1 до 3 (16 градаций серого, 16x16, см. рис. 3). Примените вероятностную нейронную сеть для решения той же проблеммы (Probabilistic neural networks, newpHC). Сравните качество работы обеих сетей на зашумленных образцах (для наложения шума прибавьте к каждому пикселю случайное число).

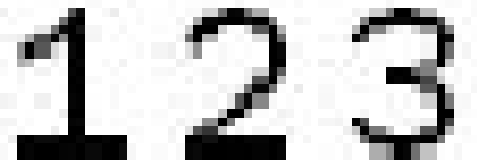

Рис. 3. Изображения цифр для распознавания.

4. Построить сеть линейного слоя (Linear layer, newlin) для восстановления последовательности $f(n)=n\times 2^n+3^n$ по выборке из 3-х предыдущих значений без использования временных задержек. Сравните коэффициенты рекурентной формулы, полученные при обучении сети, с их точными значениями. Постройте сеть линейного слоя с использованием временных задержек (3-й аргумент в функции newlin) для восстановления той же последовательности.
5. Построить множество точек

```
t = 0:0.05:(6*pi);
n  =length(t);
P = zeros(2,n);
r = exp(2:(3/n):(5-3/n))/exp(5)*50;
P(1,:) = r.*cos(t)+rand(1,n);
P(2,:) = r.*sin(t)+rand(1,n);
```

Полученное множество должно выглядеть как на рис. 4.

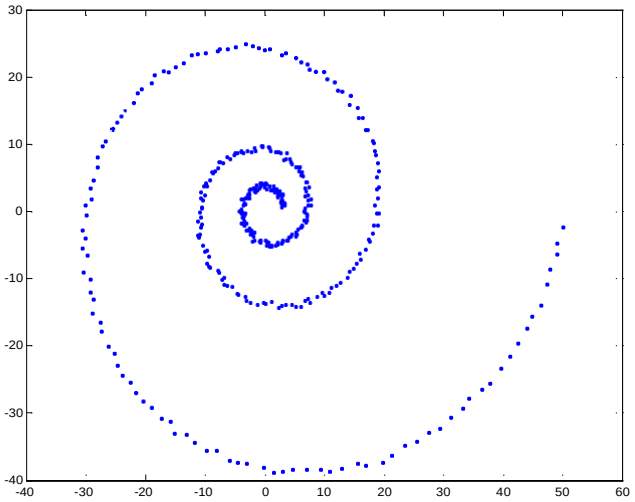

Рис. 4. Множество точек для обучения сети по правилу Кохонена.

Построить соревновательный слой (Competitive layer, newc) из 25 нейронов и обучить его на данном множестве. Показать траектории движения нейронов при обучении и как будут располагаться нейроны после его окончания. Обучить самоорганизующуюся карту (Self-organizing map, newsom) (кол-во нейронов то же) на том же множестве. Сравнить расположение нейронов в первом и втором случае. Объяснить полученные результаты.

6. Сравнение применения сети прямого распространения (Feed-forward, newff) и сети Хопфилда (Hopfield, newhop) для распознавания изображений букв (черно-белые изображения, 20x20, см. рис. 5). Оцените качество работы сетей на зашумленных образцах (для наложения шума случайным образом инвертируйте несколько пикселей).

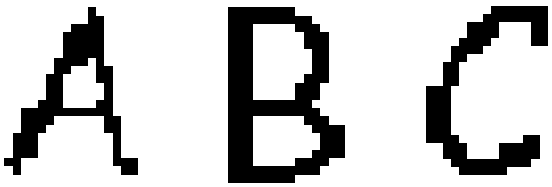

Рис. 5. Изображения букв для распознавания.

7. Применение саморганизующейся карты (Self-organizing map (SOM), newsom) для сжатия изображения. В качестве изображения взять bmp-файл (256 градаций серого) и разбить его на кадры nxn. Преобразовать последовательность кадров изображения в последовательность n^2-элементных векторов и использовать для обучения сети SOM (желательно взять 256 нейронов в сети). Процесс сжатия заключается в преобразовании последовательности кадров изображения в последовательность чисел (номер класса, к которому был отнесен данный кадр). Восстановление изображения заключается в замене каждого числа вектором, представляющим “центр тяжести” данного класса (он сохраняется во входных весах сети). Пример работы данного механизма приведен на рис. 6. Привести пример работы построенной сети для другого изображения. Оценить степень сжатия данных при помощи такого механизма.

Рис. 6. Пример применения сети SOM для сжатия изображения (исходное изображение слева, а восстановленное - справа).

8. Сжатие изображение при помощи выделения главных компонент. Для выделения компонент применить двухслойную сеть прямого распространения (Feed-forward, newff), в которой размер первого слоя совпадает с количеством компонент K, а второго (выходного) слоя – с входным слоем. Разбить изображение на кадры 10x10, в качестве входных векторов сети можно использовать либо строки, либо столбцы матрицы пискселей одного кадра. Процесс сжатия заключается в получении для каждого кадра матриц разложения и реконструкции (сохраняемых в весах сети входного и второго слоев соответственно) и последующей замене каждого столбца (строки) $\boldsymbol{x}$ вектором $\boldsymbol{y}$ из K главных компонент. Т.о. каждый кадр будет заменен на матрицу рекострукции и последовательность векторов $\boldsymbol{y}$. Пример работы данного механизма приведен на рис. 7. Оценить степень сжатия данных при помощи такого механизма.

Рис. 7. Пример сжатия изображения при помощи выделения главных компонент (слева-направо: исходное изображение, восстановленное при использовании 1-ой главной компоненты, при использовании 4-х главных компонент).

9. Примененить сеть прямого распространения (Feed-forward, newff) для распознавания характерного участка на графике регистрации некоторого процесса при помощи перемещаемого временного окна (рис. 8, распознаваемый момент отмечен пунктирной линией). Как будет вести себя сеть при анализе искаженного сигнала (изменение постоянной составляющей сигнала (сдвиг вверх-вниз), его сжатие-растяжение по вертикали, наложение шума)? Дополниттельное задание - попытаться применить сеть Эльмана (newelm) для решения той же задачи.

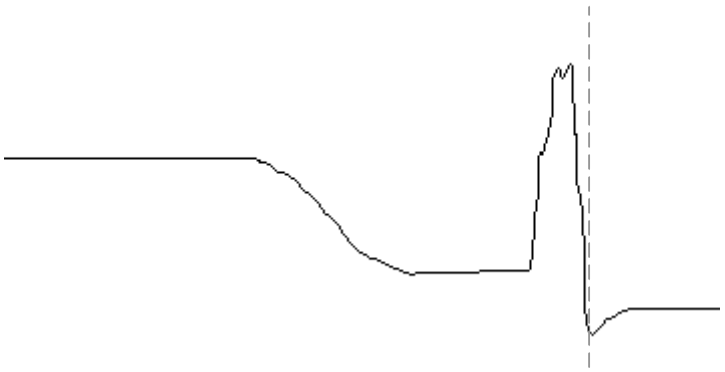

Рис. 8. Пример графика процесса регистрации некоторого процесса.

10. Применить сеть прямого распространения (Feed-forward, newff) для прогнозирования значений временного ряда методом погружения в N-мерное пространство. Временной ряд задан формулами:

```
t = 0:0.1:8;
n = length(t);
 y = sin(t/5)+rand(1,n)/10-0.05;
```

Данный временной ряд приведен на рис. 9. Сеть должна иметь N входов, M нейронов в первом слое с тангенсальной функцией активации (tansig) и одним нейроном в выходном слое с линейной функцией активации (purelin). Построить поверхность ошибки прогноза в зависимости от N и от M, а также график зависимости средней величины ошибки от периода прогнозирования.

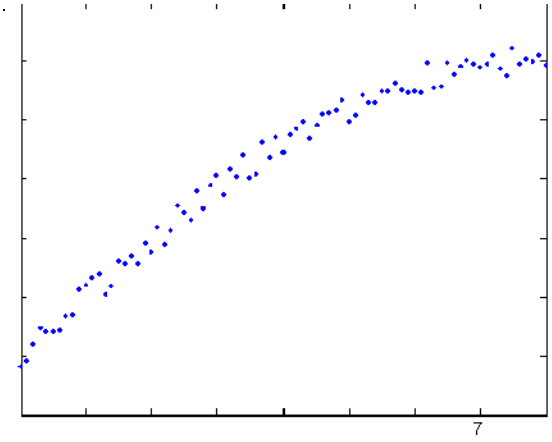

Рис. 9. Временной ряд для построения прогноза.

В отчете должны присутствовать:

1. Описание способа решения поставленной задачи.
2. Описание использованных команд и функций.
3. Графики, иллюстрирующие процесс решения и полученный результат.

Краткие теоретические сведения

Нейросетевой инструментарий среды MATLAB (Neural Net Toolbox) – универсальная нейронная сетевая среда. В пакет включено практически все, что касается процедур создания нейронных сетей и анализа полученных результатов, в частотности, их интерпретации.

Все команды MATLAB принимают наличие сетевого объекта, который называется “сеть” (net). Для того чтобы создать такой объект, необходимо выполнить команду

```
net = network;
```

которая дает чистую сеть, то есть без “особенностей” (характеристик), которые определяют специфику задачи. В качестве таких особенностей в первую очередь выступает конфигурация сети.

Сеть в MATLAB состоит из слоев, которые делятся на входные и скрытые. Входные слои служат для подачи на вход сети входных векторов и имеют лишь одну характеристику - размер, соответствующий размерности входных векторов. Скрытые слои имеют размер, соответствующий количеству нейронов, и функцию активации. Каждый скрытый слой может быть соединен с любым из входных слоев, а также с любым другим скрытым слоем. Кроме того каждый скрытый слой может быть выходным или целевым. Выходные слои определяют выход сети, а выходы целевых слоев используются при обучении сети с учителем. Таким образом, MATLAB позволяет создавать сети с довольно сложной конфигурацией, однако на практике как правило используются многослойные сети, в которых каждый скрытый слой соединен с одним следующим за ним слоем, первый скрытый слой соединен с единственным входным слоем, а последний скрытый слой назначается и выходным, и целевым.

Для создания нейронной сети с заданной конфигурацией можно использовать расширенный синтаксис команды *network*:

```
net = network(numInputs, numLayers, biasCoHCect, inputCoHCect,
layerCoHCect, outputCoHCect, targetCoHCect),
```

где

- *numInputs* – количество входных слоев.
- *numLayers* – количество скрытых слоев.
- *biasCoHCect* – вектор *numLayers x 1*, состоящий из нулей и единиц. Если *i*-ая компонента данного вектора равна 1, то *i*-ый скрытый слой имеет смещение.
- *inputCoHCect* - матрица *numLayers x numInputs*, также состоящая из нулей и единиц. Если *inputCoHCect(i, j) = 1*, то *i*-ый скрытый слой соединен с *j*-ым входным слоем.
- *layerCoHCect* – матрица *numLayers x numLayers*, определяющая соединения между скрытыми слоями. Если *layerCoHCect(i, j) = 1*, то имеется связь из *j*-ого скрытого слоя в *i*-ый скрытый слой.
- *outputCoHCect* – вектор *1 x numLayers*, определяющий, какие скрытые слои являются выходными. Если *outputCoHCect(i) = 1*, то *i*-ый скрытый слой является выходным.
- *targetCoHCect* - вектор *1 x numLayers*, определяющий, какие скрытые слои являются целевыми. Если *targetCoHCect(i) = 1*, то *i*-ый скрытый слой является целевым.

Созданный сетевой объект имеет ряд свойств. Во-первых, это свойства *net.numInputs, net.numInputs, net.numLayers, net.biasCoHCect, net.inputCoHCect, net.layerCoHCect, net.outputCoHCect, net.targetCoHCect*, аналогичные соответствующим параметрам создания сетевого объекта.

Во-вторых, это свойства, определяющие весовые коэффициенты:

- *net.IW{i,j}* – матрица весовых коэффициентов для связи *j*-го входного слоя с *i*-ым скрытым слоем.
- *net.LW{i,j}* - матрица весовых коэффициентов для связи *j*-го скрытого слоя с *i*-ым скрытым слоем.
- *net.b{i}* – величины смещений для *i*-го скрытого слоя.

В-третьих, это свойства, содержащие подъобъекты, определяющие некоторые дополнительные параметры:

- *net.inputs{i}* – структура, содержащая параметры *i*-го входного слоя.

  Для каждого входного слоя можно задать размерность векторов, поступающих на его вход, через свойство *net.inputs{i}.size*, а также входной диапазон через свойство *net.inputs{i}.range*. Для задания входного диапазона используется матрица

net.inputs{i}.size x 2. В *k*-ой строке этой матрицы указывается минимальное и максимальное значения, которые может принимать *k*-ая компонента входного вектора.

- *net.layers{i}* – структура, содержащая параметры *i*-го скрытого слоя.

  Для каждого скрытого слоя можно задать количество нейронов в нем через свойство *net.layers{i}.size*, а также функцию активации через свойство *net.layers{i}.transferFcn*, в качестве которой можно указать одну из следующих функций:

  - *logsig* – униполярная сигмоида: *logsig(x) = 1 / (1 + exp(-x)*.
  - *tansig* – биполярная сигмоида: *tansig(x) = 2/(1+exp(-2*x))-1*.
  - *purelin* – линейная функция: *purelin(x) = x*.
  - *satlin* – ассиметричная линейная функция с насыщением:

$$satlin(x)=\begin{cases}0, & x \le 0\\ x, & 0 \le x \le 1\\ 1, & 1 \le x\end{cases}$$

  - *satlins* – симетричная линейная функция с насыщением:

$$satlins(x)=\begin{cases}-1, & x \le -1\\ x, & -1 \le x \le 1\\ 1, & 1 \le x\end{cases}$$

  - *hardlim* – ассиметричный скачок:

$$hardlim(x)=\begin{cases}1, & x \ge 0\\ 0, & x \le 0\end{cases}$$

  - *hardlims* – симметричный скачок:

$$hardlims(x)=\begin{cases}1, & x \ge 0\\ -1, & x \le 0\end{cases}$$

- *net.biases{i}* – структура, содержащая параметры смещений *i*-го скрытого слоя.
- *net.inputWeights{i,j}* – структура, содержащая параметры связи из *j*-го входного слоя в *i*-ый скрытый слой. Наиболее важным параметром здесь является весовая функция *net.inputWeights{i,j}. weightFcn*, определяющая каким образом используется матрица весов при вычислении выходов нейронов. В качестве весовой функции может выступать:
  - *dotprod* –произведение.
  - *normprod* – нормализованное произведение
  - *dist* – евклидово расстояние.
  - *negdist* – евклидово расстояние, взятое с противоположным знаком.
  - *mandist* – Манхэттновское расстояние
- *net.layerWeights{i,j}* – структура, содержащая параметры связи из *j*-го скрытого слоя в *i*-ый скрытый слой. При помощи данной структуры также имеется возможность задать весовую функцию при помощи свойства *net.layerWeights{i,j}.weightFcn*.
- *net.outputs{i}* – структура, содержащая параметры *i*-го выходного слоя.
- *net.targets{i}* – структура, содержащая параметры *i*-го целевого слоя.

Для просмотра значений простых свойств или состава сложных свойств достаточно ввести их в командную строку MATLAB и нажать ENTER. Кроме того в MATLAB имеется та-

кое средство как Simulink, способное построить визуальное представление нейронной сети. Для того чтобы воспользоваться этой возможностью достаточно выполнить команду

```
gensim(net);
```

Ниже приведен код, создающий сеть с одним входным слоем и двумя скрытыми слоями:

```
net = network;
net.numInputs = 1;
net.numLayers = 2;
```

То же самое можно сделать следующим образом:

```
net = network(1,2);
```

Ниже приведен код, создающий сеть, в которой входной слой соединен с первым скрытым слоем, первый скрытый слой соединен со вторым скрытым слоем, который является выходным и целевым:

```
net = network(1,2,[1;0],[1; 0],[0 0; 1 0],[0 1],[0 1]);
```

При этов в созданной сети первый скрытый слой имеет смещение, а второй скрытый слой – нет.

Инициализация

В MATLAB есть две возможности для обучения нейронных сетей: тренировка и адаптация. При тренировке веса сети обновляются после предъявления ей всего множества входных векторов. После этого полученные значения ошибок усредняются в соответствии с выбранной *функцией выполнения – net.performFcn*. В качестве ее значения можно указать одну из следующих стандартных функций:

- *mse* – среднеквадратичная ошибка;
- *mae* – среднеабсолютная ошибка.

Далее веса сети обновляются в соответствии с выбранной тренировочной функцией - *net.trainFcn*. В среде MATLAB имеется большой выбор таких функций:

- *traingd* – обучение по алгоритму наискорейшего спуска;
- *traingdm* – по алгоритму наискорейшего спуска с учетом момента;
- *trainlm* - по алгоритму Левенберга-Марквардта;
- *trainrp* – по алгоритму RPROP и др.

Один такой цикл обучения называется эпохой. Дополнительные параметры обучения устанавливаются через структуру *net.trainParam*. Состав данной структуры зависит от выбранной тренировочной функции. Общими и наиболее часто используемыми являются следующие параметры:

- *net.trainParam.lr* – коэффициент обучения;
- *net.trainParam.goal* – максимально допустимое значение ошибки (цель обучения);
- *net.trainParam.epochs* – максимальное количество эпох обучения;
- *net.trainParam.show* – определяет промежуток времени в эпохах между сообщениями о статусе процесса обучения.

Сам процесс тренировки вызывается при помощи функции *train*:

```
P = [0.3 0.2 0.54 0.6;
     1.2 2.0 1.4  1.5];
T = [0 1 1 0];
net = train(net, P, T);
```

В этом примере *P* – матрица, представляющая множество входных данных, каждый столбец которой интерпретируется как один из входных векторов. А *T* – это матрица целевых значений, каждый столбец которой интерпретируется как целевой вектор. При этом количество столбцов в матрицач *P* и *T* должно совпадать.

В случае обучения без учителя матрица T не указывается, т.к. обучение происходит лишь на основе входных векторов.

```
net = train(net, P);
```

Адаптация вызывается при помощи функции *adapt*. При адаптации имеется возможность контролировать порции входных векторов, после подачи которых будет происходить обновление весов:

```
P = {[0.3; 1.2] [0.2; 2.0] [0.54; 1.4] [0.6; 1.5]}
T = {[0] [1] [1] [0]};
net = adapt(net, P, T);
```

В приведенном примере обновление весов будет происходить после подачи каждого из входных векторов. Адаптация весов происходит в соответствии с *методом адаптации* – *net.adaptFcn*. На данный момент предусмотрен лишь один метод адаптации – *adaptwb*, в соответствии с которым используются функции обучения, устанавливаемые для всех весов и смещений в отдельности через свойства *net.inputWeights{i,j}.learnFcn*, *net.layerWeights{i,j}.learnFcn* и *net.biases{i}.learnFcn*. В качестве обучающей функции можно указать

- *learnp* – обучение по правилу персептрона;
- *learngd* – по алгоритму наискорейшего спуска;
- *learngdm* – по алгоритму наискорейшего спуска с учетом момента инерции;
- *learnh* – по правилу Хебба;
- *learnlv1* – методом квантования векторов;
- *learnis* – по правилу обучения инстара;
- *learnos* – по правилу обучения аутстара и др.

Количество проходов по всему множеству обучающих данных устанавливается в свойстве *net.adaptParam.passes*.

После обучения сети можно подать на ее вход множество входных векторов и получить множество выходных векторов при помощи функции *sim*:

```
Y = sim(net, P);
```

В этом случае, как и при обучении, каждый столбец матрицы *P* будет соответствовать одному входному вектору, а каждый столбец матрицы *Y* – одному выходному вектору.

Таким образом общая схема работы с нейронными сетями в среде MATLAB сводится к следующей схеме:

1. Создание нейросетевого объекта и его инициализация.
2. Обучение нейронной сети.
3. Получение значений выходов нейронной сети и их интерпретация.

Следует отметить, что в среде MATLAB имеется возможность использовать для создания “типичного” нейросетевого объекта одну из стандартных функций. Далее будут рассмотрены некоторые наиболее часто используемых из них.

1. ***Сеть прямого распространения (newff)***.

   Данная сеть является обычной многослойной сетью с одним входным слоем и несколькими скрытыми слоями, последовательно соединенными друг с другом. Последний скрытый слой назначается выходным и целевым. Для создания данной сети используется функция *newff*:

```
net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF);
```

где *PR* - матрица *R* x 2 минимальных и максимальных векторов для *R* входных элементов; *Si* - размер *i*-ого скрытого слоя; *TFi* – функция активации *i*-ого скрытого слоя; *BTF* – тренировочная функция; *BLF* – обучающая функция, используемая при адаптации; *PF* – функция выполнения. Только первые два параметра являются обязательными.

Ниже приведен пример использования данной сети для аппроксимации модельной функции MATLAB *humps*:

```
x = -1:0.1:3; y = humps(x);
net = newff(minmax(x), [5 1], ...
            {'tansig' 'purelin'}, ...
            'trainlm', 'learngdm', 'mse');
net.trainParam.epochs = 100;
net = train(net, x, y);
a = sim(net, x);
plot(x, a, x, a-y);
```

Ниже приведен пример использования данной сети для аппроксимации поверхности *z = cos(x)*sin(y)*:

```
x = -2:0.1:2; y = -2:0.1:2; z = cos(x)'*sin(y);
P = [x; y]; T = z;
net = newff(minmax(P), [25 length(x)], ...
            {'tansig' 'purelin'}, ...
            'trainrp', 'learngdm', 'mse');
net.trainParam.epochs = 1000;
net = train(net, P, T);
Y = sim(net, P);
mesh(x, y, Y);
```

2. ***Радиальная сеть (newrb)***.

Данная сеть состоит из двух слоев. Первый слой состоит из нейронов, вычисляющих радиальную функцию *RADBAS* как функцию от расстояния *DIST* между позицией нейрона и входным вектором. Позиции нейронов сохраняются в строках матрицы весов *net.IW{1,1}*. Второй слой состоит из обычных нейронов с линейной функцией активации *PURELIN*. При создании данной сети подбирается количество нейронов в первом слое. Сначала берется один нейрон и вычисляются веса второго слоя. Если получившаяся ошибка больше допустимой, то добавляются нейроны в первый слой на позиции тех входных векторов, для которых ошибка достигает своего максимального значения, и процесс повторяется.

Для создания данной сети используется функция *newrb*:

```
net = newrb(P,T,GOAL,SPREAD,MN,DF)
```

где *P* – матрица *R x Q* из *Q* входных векторов размерностью *R*, *T* – матрица *S x Q* из *Q* целевых векторов размерностью *S*, *GOAL* – максимально допустимая ошибка, *SPREAD* – параметр сглаженности радиальных функций, *MN* – максимально допустимое количество нейронов в первом слое, *DF* – количество нейронов, добавляемых на каждом шаге. Лишь первые два параметра являются обязательными.

Ниже приведен пример использования данной сети для аппроксимации поверхности *z = cos(x*sin(y))*:

```
x = -2:0.1:2; y = -2:0.1:2; z = cos(x'*sin(y));
P = [x; y]; T = z;
net = newrb(P, T);
Y = sim(net, P);
```

```
mesh(x, y, Y);
```

3. ***Вероятностная нейронная сеть (newpHC)***.

Данная сеть является одной из разновидностей радиальных нейронных сетей, строящихся на основе теоремы Ковера. Она используется для классификации, причем допускает только обучение с учителем. Количество нейронов в первом слое данной сети устанавливается равным количеству входных векторов. Веса первого слоя устанавливаются равными транспонированной матрице входных данных, а веса второго слоя устанавливаются такими, чтобы обеспечить желаемый выход сети.

Для создания данной нейронной сети используется функция *newpHC*:

```
net = newpHC(P,T,SPREAD)
```

где *P* – матрица, столбцы которой составляют множество входных векторов, *T* – матрица, столбцы которой составляют желаемые выходные вектора, *SPREAD* - параметр сглаженности радиальных базисных функций (в качестве таковой здесь используется функция MATLAB *RADBAS*).

Ниже приведен пример использования данной нейронной сети:

```
%  Входные  значения
P = [1 2 3 4 5 6 7];
% Соответствующие классы
Tc = [1 2 3 2 2 3 1];
% Преобразование номеров классов в вектора
T = ind2vec(Tc)
% Создание нейронной сети и ее обучение
net = newpHC(P,T);
% Проверка работы нейронной сети
Y = sim(net,P)
% Преобразование полученных векторных данных в номера классов
Yc = vec2ind(Y)
```

4. ***Сеть соревновательного слоя (newc)***.

Данная сеть состоит из одного слоя нейронов, обучаемых по правилу Кохонена со стратегией WTA (WiHCer Takes All). Позиции нейронов сохраняются в строках матрицы *net.IW{1,1}*. На выходе такой сети будет набор векторов, состоящих из нулей и единиц. В каждом из них присутствует лишь одна единица, показывающая, какой из нейронов победил.

Для создания данной сети используется функция *newc*:

```
net = newc(PR,S,KLR,CLR)
```

где *PR* - матрица *R x 2* минимальных и максимальных значений для *R* компонентов входных векторов; *S* - количество нейронов, *KLR* – коэффициент обучения по правилу Кохонена, *CLR* – коэффициент “совести”, используемый для повышения шансов на победу тех нейронов, которые побеждают реже других, таким образом, чтобы все нейроны побеждали с примерно одинаковой частотой. Лишь первые два параметра являются обязательными.

Рассмотрим пример обучения такой сети на некотором множестве точек:

```
n = 100;
P = [[ 9+5*rand(1,n);  9+5*rand(1,n)]
     [19+5*rand(1,n); 19+5*rand(1,n)]]
plot(P(1,:), P(2,:), '.')

net = newc(minmax(P), 2);
```

```
net = train(net,P);

Y = sim(net,P);
Yc = vec2ind(Y);

figure; hold;
for i=1:length(P)
  if Yc(i) == 1
    c = '.r';
  else
    c = '.b';
  end
  plot(P(1,i), P(2,i), c);
end
```

В приведенном примере было создан массив точек, состоящий из двух кластеров. На этих точках была обучена сеть соревновательного слоя с двумя нейронами. Далее при помощи данной сети множество точек было разделено на два класса, а полученные результаты были выведены на графике: точки, более близкие к первому нейрону, были выведены красным цветом, а точки, более близкие ко второму нейрону – синим цветом. Сами позиции нейронов можно вывести при помощи команды

```
plotsom(net.IW{1,1}');
```

5. Самоорганизующаяся карта (newsom).

Данная сеть является дальнейшим развитием сети соревновательного слоя. Она также состоит из одного слоя нейронов, но в ней на каждом шаге обучению подвергается не только победивший нейрон, но и несколько ближайших к нему нейронов. Процесс обучения состоит из двух этапов: фазы упорядочивания и фазы настройки. На первом этапе нейроны должны примерно равномерно распределиться по всему множеству входных точек. Для этого на первом шаге обучению подвергаются все нейроны, в процессе обучения количество обучаемых нейронов уменьшается, на последнем шаге обучению подвергается лишь один победивший нейрон. На втором этапе нейроны должны определить все кластеры, составляющие обучаемое множество. На этом этапе обучению подвергается победивший нейрон и нейроны, находящиеся к нему ближе заданного расстояния. Кроме того имеется возможность управлять начальным расположением нейронов в пространстве.

Для создания данной сети используется функция *newsom*:

```
net = newsom(PR,[D1,D2,...,DNl],TFCN,DFCN,OLR,OSTEPS,TLR,TND)
```

где *PR* - матрица *R x 2* минимальных и максимальных значений для *R* компонентов входных векторов; нейроны сети будут равномерно распределены в *Nl*-мерном пространстве по *Di* слоев в каждом измерении (например, [5 8] задает расположение 40 нейронов в двумерном пространстве в узлах решетки 5 x 8); *TFCN* – функция топологии, определяющая, каким именно образом будут расположены нейроны (*gridtop* – в узлах прямоугольной решетки, *hextop* – в узлах решетки, состоящей из шестиугольников); *DFCN* – используемая функция расстояния (*dist* – евклидово расстояние, *mandist* – манхеттоновское расстояние); *OLR* – коэффициент обучения в фазе упорядочивания, *OSTEPS* – количество шагов, отводимых по фазу упорядочивания; *TLR* – коэффициент обучения в фазе настройки (обычно *TLR* << *OLR*); *TND* – расстояние от победившего нейрона, в пределах которого подвергаются обучению другие нейроны во время второй фазы.

Ниже приведен пример использования данной сети:

```
P = [];
for i = 1:100
  P = [P [9+5*mod(i,4)+5*rand; 9+5*mod(i,4)+5*rand]];
end
plot(P(1,:), P(2,:), '.'); hold;

net = newsom(minmax(P), [2 2], 'hextop', 'dist', 1, 100, 0.1, 0);
plotsom(net.layers{1}.positions);

net.trainParam.epochs = 500;
net = train(net,P);

Y = sim(net,P);
Yc = vec2ind(Y);

figure; hold; colors = ['b' 'g' 'r' 'c'];
for i=1:length(P)
  plot(P(1,i), P(2,i), ['.' colors(Yc(i))]);
end
plotsom(net.iw{1,1},net.layers{1}.distances);
```

6. Сеть квантования векторов (newlvq).

Данная сеть также является дальнейшим развитием сети соревновательного слоя. Она также предназначена для разделения входных векторов на классы, но в отличие от соревновательного слоя может обучаться только с учителем. Основным преимуществом данной сети является то, что одному классу может соответствовать не один, а несколько нейронов. Она состоит из двух слоев: первый слой является обычным соревновательным слоем, а второй состоит из обычных линейных нейронов и используется для объединения сигналов от всех нейронов, соответствующих одному классу. Выход данной сети, как и выход соревновательного слоя, состоит из нулей и одной единицы, показывающей, к какому классу принадлежит входной вектор.

Для создания данной сети используется функция *newlvq*:

```
net = newlvq(PR,S1,PC,LR,LF)
```

где *PR* - матрица *R x 2* минимальных и максимальных значений для *R* компонентов входных векторов; *S1* – количество нейронов в первом слое; *PC* – вектор [p1 p2 … pn] процентных соотношений, показывающих, какая часть нейронов первого слоя соответствует данному классу (сумма всех pi должна быть равна 1); *LR* – коэффициент обучения, *LF* – обучающая функция (должна быть равна *learnlv1*, еще один алгоритм – *learnlv2* – применяется для “тонкой” настройки сетей, уже обученных по алгоритму *learnlv1*).

Ниже приведен пример применения данной сети для разделения двух классов точек, которые нельзя разделить при помощи сети соревновательного слоя:

```
P = []; Tc = []; colors = ['r' 'b'];
figure; hold;
for i = 1:100
  P = [P [9+5*mod(i,4)+5*rand; 9+5*mod(i,4)+5*rand]];
  Tc = [Tc 2-mod(i,2)];
  plot(P(1,i), P(2,i), ['.' colors(Tc(i))]);
end
T = ind2vec(Tc);
```

```
net = newlvq(minmax(P), 4, [0.5 0.5]);
plotsom(net.iw{1,1}');

net.trainParam.epochs = 500;
net = train(net,P,T);

Y = sim(net,P);
Yc = vec2ind(Y);

figure; hold;
for i=1:length(P)
  plot(P(1,i), P(2,i), ['.' colors(Yc(i))]);
end
plotsom(net.iw{1,1}');
```

7. Сеть Хопфилда (newhop).

Сеть Хопфилда используется для создания автоассоциативной памяти. С ее помощью можно запомнить набор векторов, состоящих из +1 и -1. При подачи на ее вход некоторого вектора на выходе сети будет получен один из запомненных ее векторов, который сеть посчитала наиболее похожим на входной. В MATLAB реализована модифицированная сеть Хопфилда, обладающая меньшей емкостью, но лучше запоминающая вектора. Кроме того, при восстановлении векторов она допускает подачу на вход векторов из произвольных значений (а не только +1 и -1).

Для создания данной сети используется функция *newhop*:

```
net = newhop(T)
```

где *T* – матрица, каждый столбец которой интерпретируется как запоминаемый вектор.

Ниже приведен пример использования данной сети для запоминания двух точек в трехмерном пространстве:

```
T = [-1 -1 1; 1 -1 1]';
net = newhop(T);
```

Далее проверим работу данной сети:

```
Y = sim(net, 2, [], T);
```

Обратите внимание на необычный синтаксис команды *sim*. Это связано с тем, что у сети Хопфилда фактически нет входного слоя. Вместо этого задается начальное состояние единственного скрытого слоя нейронов. В этом случае указывается количество векторов и матрица, составленная из них.

Проверим работу сети на искаженных векторах:

```
T = [0.7 0.6 -0.25]';
Y = sim(net, 1, [], T);
```

В этом случае на выходе сети будет получен вектор, отличный от запомненных. Это связано с тем, что для восстановления значений векторов сети Хопфилда требуется несколько циклов работы, а в данном случае проведен лишь один цикл. Для того чтобы провести несколько циклов (в данном случае 5) используется следующий синтаксис:

```
T = {[0.7; 0.6; -0.25]};
Y = sim(net, {1 5}, {}, T);
```

После этого *Y{i}* будет содержать выходной вектор сети Хопфилда после *i*-ого цикла работы.

8. Линейный слой (newlin).

Линейный слой состоит из одного слоя обычных нейронов с линейной функцией активации.

Для создания данной сети используется функция *newlin*:

```
net = newlin(PR,S,ID,LR)
```

где *PR* - матрица *R x 2* минимальных и максимальных значений для *R* компонентов входных векторов; *S* – количество нейронов; *ID* – вектор возрастающих неотрицательных временных задержек, задающий от каких входных векторов поступает сигнал вход линейного слоя (например, [0] – сигнал поступает от входного вектора, [0 1] – сигнал поступает от входного вектора и от предыдущего входного вектора, [0 3 5] - сигнал поступает от входных векторов текущего и поступавших 3 и 5 шагов назад; в текущей реализации наличие 0 в первой позиции является обязательным); *LR* - коэффициент обучения. Лишь первые два параметра являются обязательными.

Ниже приведен пример сети для восстановления синусоидального сигнала длительностью 5 секунд, измерения которого выполнены по выборке по норме 40 образцов в секунду, по 5 предыдущим значениям:

```
time = 0:0.025:5; y = sin(time*4*pi); Q = length(y);
P = {}; T = {}; P1 = [];
for i = 1:Q-5
  P = [P {[y(i) y(i+1) y(i+2) y(i+3) y(i+4)]'}];
  P1 = [P1 P{i}];
  T = [T {y(i+5)}];
end;

net = newlin([-ones(5,1) ones(5,1)], 1);
net.inputWeights{1.1}.learnFcn = 'learngdm';
net.adaptParam.passes = 10;
net = adapt(net, P, T);
a = sim(net, P1);
plot(a);
```

Обратите внимание, что здесь использовалась функция *adapt*, а не *train*, как в других примерах. Это связано с тем, что в случае использования тренировки ошибка будет сначала усредняться по всем входным значениям, а потом использоваться для обучения, что не принесет пользу в силу периодичности значений синусоиды (положительные значения будут компенсировать отрицательные).

Тот же пример с использованием временных задержек будет выглядеть следующим образом:

```
time = 0:0.025:5; y = sin(time*4*pi); Q = length(y);
T1 = {}; T2 = {};
for i = 1:Q-1
  T1 = [T1 {y(i)}];
  T2 = [T2 {y(i+1)}];
end;

net = newlin([-1 1], 1, [0 1 2 3 4]);
net.inputWeights{1.1}.learnFcn = 'learngdm';
net.adaptParam.passes = 10;
net = adapt(net, T1, T2);
a = sim(net, y);
```

Примечание

Для работы с изображениями применяются следующие функции:

- **`A = imread(filename);`**

  В результате в матрицу *A* будет занесены значения цветов пикселей. Размеры загруженного изображения можно узнать следующим образом:

  `[height width] = size(A)`**`imshow(A);`** В результате появится окно с изображением, хранящимся в матрице *A*.

- **`imwrite(A, filename);`**

  В результате изображение, хранящееся в матрице *A*, будет сохранено в файле с указанным именем.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Основная литература

1. Сидоркина И. Г. Системы искусственного интеллекта. - Москва: Кнорус, 2011. - 248 с.
2. Павлов С. И. Системы искусственного интеллекта: учебное пособие, Ч. 1 Томск: Томский государственный университет систем управления и радиоэлектроники, 2011.-175с. [ЭБС «Университетская библиотека онлайн»];

Дополнительная литература:

1. Павлов С. И. Системы искусственного интеллекта: учебное пособие, Ч. 2 Томск: Томский государственный университет систем управления и радиоэлектроники, 2011.-194с. [ЭБС «Университетская библиотека онлайн»]

Нелин В.М.

СЕТИ И ТЕЛЕКОММУНИКАЦИИ

методические указания по выполнению и оформлению лабораторных работ для бакалавров направления подготовки 09.03.01 - Информатика и вычислительная техника

Отпечатано в типографии
Северо-Кавказского института бизнеса, инженерных и информационных технологий
г. Армавир, ул. Дзержинского 62/1